Counting Instances of Software Components

Marc Bezem and Hoang Truong

Department of Informatics, University of Bergen,
PB.7800, N-5020 Bergen, Norway
{bezem,hoang}@ii.uib.no

Abstract. Component software is software that has been assembled from various pieces of standardized, reusable computer programs, so-called components. Executing component software creates instances of these components. For several reasons, for example, limited resources and/or application requirements, it can be important to have control over the number of such instances. Clearly, in cases where this is possible, design-time or compile-time control is to be preferred to run-time control. We give an abstract component language and a type system which ensures that the number of simultaneously active instances of any component never exceeds a (sharp) bound expressed in the type. The language features instantiation and reuse of components, as well as sequential composition, choice and scope. Alternatively one can view the expressions in the language as denoting processes where the atomic actions are interpreted as either creating new, or reusing old instances.

1 Introduction

Component software is built from various components, possibly developed by third-parties [11], [14]. These third-party components may in turn use other components. Upon execution instances of all these components are created. The process of creating an instance of a component c does not only mean the allocation of memory space for c's code and data structures, the creation of instances of c's subcomponents (and so on), but possibly also the binding of other hardware resources. In many cases, resources are limited and components are bound to have only a certain number of simultaneously active instances. For example, a serial output device can usually stand only one instance of a driver-component, serialized ID generators should be unique [7], [6]. Most servers can have only a certain number of clients.

When building component software it can easily happen that, unforseen by the developer, different instances of the same component are created. Creating more active instances than allowed can lead to errors. There are several ways to meet this challenge, ranging from testing to dynamic instantiation schemes. Type systems have traditionally been used for compile-time error-checking, cf. [3], and recently for certifying important security properties, such as performance safety [5] and memory safety [4]. In component software, typing has been studied in relation to integrating components such as type-safe composition [13] or type-safe evolution [10]. In this paper we explore the possibility of a type system which

allows one to detect *statically*, at development/composition time, whether or not the number of simultaneously active instances of specific components exceeds the allowed number.

For this purpose we have designed a component language where we have abstracted away many aspects of components and have kept only those that are relevant to instantiation.[1] The main features we have retained are instantiation and reuse, sequential composition, choice and scope. Reusing a component means here to use an existing instance of the component if there is already one, and to create a new instance only if there exists none. Alternatively one can view the expressions in the language as denoting processes where the atomic actions are interpreted as either creating new, or reusing old instances. Though abstract, the strength of the primitives for composition is considerable. Choice allows us to model both conditionals and non-determinism (due to, e.g., user input). Scope is a mechanism to deallocate instances but it can also be used to model method calls. Sequential composition is associative.

This paper extends [2] in three main ways. First, we generalized the single-instance property to counting instances of components. Second, we have an additional primitive for reusing instead of always creating a new instance of a component. Third, we added a choice primitive to the language. All these aim at bringing the language closer to practice.

The paper is organized as follows. Section 2 introduces the component language with its operational semantics. In Section 3 we define types and the typing relation. Properties of the type system and the operational semantics are presented in Section 4. Last, we outline a polynomial time type inference algorithm in Section 5 and conclude in Section **??**. Technical proofs of Section 4 are delegated to the appendix.

2 A Component Language

2.1 Terms

We have two primitives (`new` and `reu`) for creating and (if possible) reusing an instance of a component, and three primitives for composition (sequential composition denoted by juxtaposition, + for choice and $\{\ldots\}$ for scope. Together with the empty expression ϵ these generate so-called *component expressions*. A *declaration* $c \prec \mathit{Exp}$ states how the component c depends on subcomponents as expressed in the component expression *Exp*. If c has no subcomponents then *Exp* is ϵ and we call c a *primitive component*. Upon instantiation or reuse of c the expression *Exp* is executed. A *component program* consist of declarations and ends with an expression which sparks off the execution, see Section 2.2.

In the formal definition below, we use extended Backus-Naur Form with the following meta-symbols: infix | for choice and overlining for Kleene closure (zero or more iterations).

[1] This should not be misunderstood as that other aspects are deemed uninteresting!

Definition 1 (Syntax). *Component programs, declarations and expressions are defined by the following syntax:*

$$
\begin{array}{lll}
\mathit{Prog} ::= \mathit{Decl}; \mathit{Exp} & & \text{(Program)} \\
\mathit{Decl} ::= \mathit{Var} \prec \mathit{Exp} \ \overline{, \mathit{Var} \prec \mathit{Exp}} & & \text{(Declarations)} \\
\mathit{Exp} \ ::= \epsilon & & \text{(Empty Expression)} \\
\quad | \ \ \mathtt{new}\, \mathit{Var} & & \text{(New Instantiation)} \\
\quad | \ \ \mathtt{reu}\, \mathit{Var} & & \text{(Reuse Instantiation)} \\
\quad | \ \ (\mathit{Exp} + \mathit{Exp}) & & \text{(Choice)} \\
\quad | \ \ \{\mathit{Exp}\} & & \text{(Scope)} \\
\quad | \ \ \mathit{Exp} \ \mathit{Exp} & & \text{(Sequential Composition)}
\end{array}
$$

We use $a, b, \ldots, z$ for component names from a set Var and $A, \ldots, E$ for expressions Exp. The following example is a well-formed component program:

$$d \prec \epsilon,\ e \prec \epsilon,\ a \prec \mathtt{new}\, d,\ b \prec (\,\mathtt{reu}\, d \{\,\mathtt{new}\, a\} + \mathtt{new}\, e\, \mathtt{new}\, a)\, \mathtt{reu}\, d; \mathtt{new}\, b\ .$$

In this example, d and e are primitive components. Component a uses one instance of component d. Component b has a choice expression before reuse of an instance of d. The first expression of the choice expression is $\mathtt{reu}\, d\{\,\mathtt{new}\, a\}$. We can view $\{\,\mathtt{new}\, a\}$ in this expression as function call $f()$ (in traditional programming languages). Function f then has body $\mathtt{new}\, a$, which means $f()$ needs a new instance of a to do its job. We abstract from the details of this job, the only relevant aspect here is that it involves a new instance of a which will be deallocated upon exiting f.

2.2 Operational Semantics

The operational semantics is defined in terms of transitions between configurations. A *configuration* is a stack ST of pairs (M, E) where M is a finite multiset over the set of component names $\mathbb{C} = \mathit{Var}$ and E is an expression. A configuration is *terminal* if it is of the form (M, ϵ), i.e. its stack contains only one pair consisting of a multiset and an empty expression. Table 1 defines the one-step transition relation $\rightsquigarrow$. As usual, $\rightsquigarrow^*$ denotes the reflective and transitive closure of $\rightsquigarrow$. Note that we can view any non-empty expression as of one of the forms: $\mathtt{new}\, xE$, $\mathtt{reu}\, xE$, $(A + B)E$ and $\{A\}E$.

We use $[ST]$ to denote the multiset of all active instances in ST, that is $[ST] = \biguplus_{j=1}^{n} M_j$ for $ST = (M_1, E_1) \circ \cdots \circ (M_n, E_n)$. The push operator $\circ$ is left associative, so that the rightmost pair (M_n, E_n) is the top of the stack. Multisets are denoted by $[\ldots]$, where sets are denoted, as usual, by $\{\ldots\}$. $M(x)$ is the multiplicity of element x in multiset M and $M(x) = 0$ if $x \notin M$. The operation $\cup$ is union of multisets: $(M \cup N)(x) = max(M(x), N(x))$. The operation $\uplus$ is additive union of multisets: $(M \uplus N)(x) = M(x) + N(x)$. We write $M + x$ for $M \uplus [x]$ and when $x \in M$ we write $M - x$ for $M - [x]$.

The example at the end of Section 2.1 is used to illustrate the operational semantics. We have two possible runs. The first one is:

(**osNew**) if $x \prec A \in \mathit{Prog}$
$ST \circ (M, \mathtt{new}\, x E) \rightsquigarrow ST \circ (M + x, AE)$

(**osReu1**) if $x \prec A \in \mathit{Prog} \quad x \notin [ST] \uplus M$
$ST \circ (M, \mathtt{reu}\, x E) \rightsquigarrow ST \circ (M + x, AE)$

(**osReu2**) if $x \prec A \in \mathit{Prog} \quad x \in [ST] \uplus M$
$ST \circ (M, \mathtt{reu}\, x E) \rightsquigarrow ST \circ (M, AE)$

(**osChoice1**)
$ST \circ (M, (A + B)E) \rightsquigarrow ST \circ (M, AE)$

(**osChoice2**)
$ST \circ (M, (A + B)E) \rightsquigarrow ST \circ (M, BE)$

(**osPush**)
$ST \circ (M, \{A\}E) \rightsquigarrow ST \circ (M, E) \circ ([], A)$

(**osPop**)
$ST \circ (M, E) \circ (M', \epsilon) \rightsquigarrow ST \circ (M, E)$

Table 1. Transition rules

$$
\begin{array}{lll}
([\,], \mathtt{new}\, b) & & \\
\quad \rightsquigarrow & ([b], (\,\mathtt{reu}\, d \{\,\mathtt{new}\, a\} + \mathtt{new}\, e\, \mathtt{new}\, a)\, \mathtt{reu}\, d) & \\
\quad \rightsquigarrow & ([b], \mathtt{reu}\, d \{\,\mathtt{new}\, a\}\, \mathtt{reu}\, d) & (\,\mathtt{reu}\, d \text{ creates a } d) \\
\quad \rightsquigarrow & ([b, d], \{\,\mathtt{new}\, a\}\, \mathtt{reu}\, d) & \\
\quad \rightsquigarrow & ([b, d], \mathtt{reu}\, d) \circ ([\,], \mathtt{new}\, a) & \\
\quad \rightsquigarrow & ([b, d], \mathtt{reu}\, d) \circ ([a], \mathtt{new}\, d) & \\
\quad \rightsquigarrow & ([b, d], \mathtt{reu}\, d) \circ ([a, d], \epsilon) & \\
\quad \rightsquigarrow & ([b, d], \mathtt{reu}\, d) & (\text{ does not create a } d) \\
\quad \rightsquigarrow & ([b, d], \epsilon) & (\text{terminal})
\end{array}
$$

The other is:

$$
\begin{array}{lll}
([\,], \mathtt{new}\, b) & & \\
\quad \rightsquigarrow & ([b], (\,\mathtt{reu}\, d \{\,\mathtt{new}\, a\} + \mathtt{new}\, e\, \mathtt{new}\, a)\, \mathtt{reu}\, d) & \\
\quad \rightsquigarrow & ([b], \mathtt{new}\, e\, \mathtt{new}\, a\, \mathtt{reu}\, d) & \\
\quad \rightsquigarrow & ([b, e], \mathtt{new}\, a\, \mathtt{reu}\, d) & \\
\quad \rightsquigarrow & ([b, e, a], \mathtt{new}\, d\, \mathtt{reu}\, d) & \\
\quad \rightsquigarrow & ([b, e, a, d], \mathtt{reu}\, d) & \\
\quad \rightsquigarrow & ([b, e, a, d], \epsilon) & (\text{terminal})
\end{array}
$$

In this example there are two possible runs and the numbers of active instances of each component are not the same during and at the end of the two runs. There are two $\mathtt{reu}\, d$'s in the first execution and only the first one creates an instance of d. The maximum for d is 2, for the others 1.

3 Type System

Let $\mathcal{R}$ be the requirement that some components in $\mathbb{C} = \mathit{Var}$ can have at most a certain number of simultaneous instances. $\mathcal{R}$ can be modelled as a total map

from $\mathbb{C}$ to $\mathbb{N} \cup \{\infty\}$. Then $\mathcal{R}(c) \in \mathbb{N}$ is the maximum allowed number of simultaneously active instances of c; $\mathcal{R}(c) = \infty$ expresses that c can have any number of instances. By convention $n < \infty$ for all $n \in \mathbb{N}$. The map $\mathcal{R}$ partitions the set of all components $\mathbb{C}$ into mutually disjoint classes $\mathbb{C}_0, \mathbb{C}_1, \ldots$ and C_∞ such that $\mathbb{C}_i = \{c \in \mathbb{C} \mid \mathcal{R}(c) = i\}$ for all $i \in \mathbb{N} \cup \{\infty\}$. Note that $\mathbb{C}_i$ may be empty for some i.

Definition 2 (Types). *Types of component expressions are tuples*

$$X = \langle X^i, X^o, X^j, X^p \rangle$$

where X^i, X^o, X^j and X^p are finite multisets over $\mathbb{C}$. We let $U, V, \ldots, Z$ range over types.

Let us first explain informally why multisets, which multisets and why four. The aim is to have a sharp upper bound of the number of simultaneously active instances of any component during the execution of the expression (X^i). Multisets are the right datastructure to collect and count such instances. In addition we want compositionality of typing, that is, we want the types to be computable from types of subexpressions. Since subexpressions may be scoped, it is necessary to have an sharp upper bound of the number of instances that are still active *after* the execution of an expression (X^o). Pairs $\langle X^i, X^o \rangle$ sufficed for the purpose of the paper [2]. Here we consider also reusing instances of components and this depends on whether there is already such an instance or not. More concretely, in a sequential composition EE' the behaviour of `reu`'s in E' depends on the instances that are active *after* the execution of E, which would violate compositionality. In order to save compositionality, we have to add two more multisets to the types, denoted by X^j, X^p. These express the same bounds as X^i, X^o, but with respect to executing the expression in a state where every component has already one active instance. Finally, we have to explain the informal phrase 'sharp upper bound'. Since we have choice, there can be different runs of the same expression, with different numbers of active instances. Now 'upper bound' means an upper bound with respect to all possible runs and 'sharp' means that, for any $c \in \mathbb{C}$, the upper bound for c is attained in at least one such run.

Based on the above intuitions, the following typings are easy:
$\texttt{new}\, d : \langle [d], [d], [d], [d] \rangle$, $\{\, \texttt{new}\, d \} : \langle [d], [\,], [d], [\,] \rangle$, $\texttt{reu}\, d : \langle [d], [d], [\,], [\,] \rangle$,
$\texttt{reu}\, d \{\, \texttt{new}\, d \} : \langle [d, d], [d], [d], [\,] \rangle$, $\texttt{reu}\, d \{\, \texttt{new}\, a \} : \langle [a, d, d], [d], [a, d], [\,] \rangle$,
where $d \prec \epsilon$ and $a \prec \texttt{new}\, d$ like in the example program in Section 2.1.

The intuitions from the above paragraph will be indispensable for understanding the typing rules later in this section, in particular the sequencing rule, but we have to prepare with some preliminary definitions.

A *basis* or an *environment* is an list of declarations: $x_1 \prec A_1, \ldots, x_n \prec A_n$ with distinct variables $x_i \neq x_j$ for all $i \neq j$, as in [1]. Let $\Gamma, \Delta, \ldots$ range over bases. When $\Gamma = x_1 \prec A_1, \ldots, x_n \prec A_n$, the set of variables $x_1, \ldots, x_n$ declared in Γ is the domain of Γ and is denoted by $Dom(\Gamma)$. A type judgment is a triple of the form

$$\Gamma \vdash_{\mathcal{R}} A : X$$

and it asserts that expression A has type X in the environment Γ, with respect to requirement $\mathcal{R}$. We ignore subscript $\mathcal{R}$ in the judgment when $\mathcal{R}$ is clear from the context or some requirement $\mathcal{R}$ is assumed in the context. We write $\vdash A : X$ if there exists a Γ such that $\Gamma \vdash A : X$.

Notation: for types X and Y, let $X \subseteq Y$, $X + Y$ and $X \cup Y$ denote *component-wise* multiset inclusion, additive union and usual union, respectively. For any expression E, let $FV(E)$ denote the set of variables occurring in E.

Definition 3 (Type judgments). *Type judgments $\Gamma \vdash A : X$ are derived by applying the typing rules in Table 2 in the usual inductive way.*

$$(\mathsf{Axiom})\frac{}{\vdash \epsilon : \langle [\,], [\,], [\,], [\,] \rangle} \qquad (\mathsf{Weaken})\frac{\Gamma \vdash A : X \quad \Gamma \vdash B : Y \quad x \notin Dom(\Gamma)}{\Gamma, x \prec B \vdash A : X}$$

$$(\mathsf{New})\frac{\Gamma \vdash A : X \quad x \notin Dom(\Gamma)}{\Gamma, x \prec A \vdash \texttt{new}\, x : \langle X^i + x, X^o + x, X^j + x, X^p + x \rangle}$$

$$(\mathsf{Reu})\frac{\Gamma \vdash A : X \quad x \notin Dom(\Gamma)}{\Gamma, x \prec A \vdash \texttt{reu}\, x : \langle X^i + x, X^o + x, X^j, X^p \rangle}$$

$$(\mathsf{Seq})\frac{\Gamma \vdash A : X \quad \Gamma \vdash B : Y \quad (X^o \uplus Y^j)(c) \leq \mathcal{R}(c) \quad A, B \neq \epsilon}{\Gamma \vdash AB : \langle X^i \cup (X^o \uplus Y^j) \cup Y^i, (X^o \uplus Y^p) \cup Y^o, X^j \cup (X^p \uplus Y^j), X^p \uplus Y^p \rangle}$$

$$(\mathsf{Choice})\frac{\Gamma \vdash A : X \quad \Gamma \vdash B : Y}{\Gamma \vdash (A + B) : X \cup Y} \qquad (\mathsf{Scope})\frac{\Gamma \vdash A : X}{\Gamma \vdash \{A\} : \langle X^i, [\,], X^j, [\,] \rangle}$$

Table 2. Typing Rules

In addition to the intuition given in the beginning of this section, some further explanation of these typing rules is in order. The rule Axiom requires no premiss and is used to take-off. The rules New and Reu allow us to type expressions $\texttt{new}\, x$ and $\texttt{reu}\, x$, respectively. The rule Weaken is used to expand bases so that we can combine typings in other rules. The side condition $x \notin Dom(\Gamma)$ prevents ambiguity and circularity. The rules Choice and Scope are easy to understand with the corresponding rules **osChoice1/2** and **osScope** of the operational semantics in mind.

The most critical rule is Seq because sequencing two expressions can lead to an increase in instances of the composed expression. Let us start with the first two multisets in the type quadruple for AB. After expression A is executed, there are at most $X^o(x)$ instances of component x. Executing B can create at most $Y^i(x)$ instances of x if x is not in X^o. Otherwise $Y^j(x)$ instances of x will be created, meaning that there are at most $((X^o \uplus Y^j) \cup Y^i)(x)$ instances of x after the execution of A and during the execution of B. Furthermore, as there are at most $X^i(x)$ instances of x created during the execution of A, the first multiset in the type of AB is the union of X^i and $((X^o \uplus Y^j) \cup Y^i)$. By a similar

reasoning we see that the surviving instances after executing AB are given by the multiset $(X^o \uplus Y^p) \cup Y^o$.

Now let us consider executing AB in a state containing at least one instance of every component. Then the maximum number of instances created by AB is clearly the maximum of those created by A and the sum of those surviving after A and those created by B, that is, $X^j \cup (X^p \uplus Y^j)$. The maximum number surviving AB is obviously the sum of those surviving A and those surviving B. This explains that the whole type for AB is $\langle X^i \cup (X^o \uplus Y^j) \cup Y^i, (X^o \uplus Y^p) \cup Y^o, X^j \cup (X^p \uplus Y^j), X^p \uplus Y^p \rangle$. We require only $X^o(x) + Y^j(x) \leq \mathcal{R}(x)$ for each $x \in \mathbb{C}_k$ in the side condition since $X^i \cup Y^i$ satisfies this bound already by induction.

Using the example in Section 2.1 with assumption that $\mathbb{C}_0 = \{c, d\}$, $\mathbb{C}_2 = \{a, b\}$, $\mathbb{C}_3 = \{e\}$, we derive type for $\texttt{new}\, b$. Note that we omitted some side conditions as they can be checked easily and we shortened the rule names. The rule Axiom is simplified. Also $\Gamma = d{\prec}\, \epsilon, a{\prec}\, \texttt{new}\, d, e{\prec}\, \epsilon$ and $\Gamma' = \Gamma, b{\prec}\, (\texttt{reu}\, d\{\texttt{new}\, a\} + \texttt{new}\, e\, \texttt{new}\, a)\, \texttt{reu}\, d$ in the following examples.

$$\textsf{Wea}\dfrac{\textsf{New}\dfrac{\vdash \epsilon : \langle [\,], [\,], [\,], [\,] \rangle}{d{\prec}\, \epsilon \vdash \texttt{reu}\, d : \langle [d], [d], [\,], [\,] \rangle} \quad \textsf{New}\dfrac{\vdash \epsilon : \langle [\,], [\,], [\,], [\,] \rangle}{d{\prec}\, \epsilon \vdash \texttt{new}\, d : \langle [d], [d], [d], [d] \rangle}}{d{\prec}\, \epsilon, a{\prec}\, \texttt{new}\, d \vdash \texttt{reu}\, d : \langle [d], [d], [\,], [\,] \rangle}$$

$$\textsf{Sco}\dfrac{\textsf{New}\dfrac{\textsf{New}\dfrac{\vdash \epsilon : \langle [\,], [\,], [\,], [\,] \rangle}{d{\prec}\, \epsilon \vdash \texttt{new}\, d : \langle [d], [d], [d], [d] \rangle}}{d{\prec}\, \epsilon, a{\prec}\, \texttt{new}\, d \vdash \texttt{new}\, a : \langle [a, d], [a, d], [a, d], [a, d] \rangle}}{d{\prec}\, \epsilon, a{\prec}\, \texttt{new}\, d \vdash \{\texttt{new}\, a\} : \langle [a, d], [\,], [a, d], [\,] \rangle}$$

Sequencing the above two derivation we have:
$d{\prec}\, \epsilon, a{\prec}\, \texttt{new}\, d \vdash \texttt{reu}\, d\{\texttt{new}\, a\} : \langle [a, d, d], [d], [a, d], [\,] \rangle$.
We can weaken the above derivation to get:
$\Gamma \vdash \texttt{reu}\, d\{\texttt{new}\, a\} : \langle [a, d, d], [d], [a, d], [\,] \rangle$ We can also derive:

$$\textsf{Seq}\dfrac{\dfrac{\ldots}{\Gamma \vdash \texttt{new}\, e : \langle [e], [e], [e], [e] \rangle} \quad \dfrac{\ldots}{\Gamma \vdash \texttt{new}\, a : \langle [a, d], [a, d], [a, d], [a, d] \rangle}}{\Gamma \vdash \texttt{new}\, e\, \texttt{new}\, a : \langle [a, d, e], [a, d, e], [a, d, e], [a, d, e] \rangle}$$

and we have: $\Gamma' \vdash \texttt{new}\, b : \langle [a, b, d, d, e], [a, b, d, e], [a, b, d, e], [a, b, d, e] \rangle$.

In this example expression $\texttt{new}\, b$ is typable. If $d \in \mathbb{C}_1$, the expression would not be typable as the side condition when sequencing $\texttt{reu}\, d$ and $\{\texttt{new}\, a\}$ would not be satisfied. Also, note that the above type derivation is not the only one but, as we will see later, the type for any expression is unique.

4 Properties

We start by giving some definitions and then state some properties of our type system. After that we will state some important properties relating types to states in the operational semantics. Proofs are delegated to Appendix A in order to improve the readability of this section.

Following [1] we fix some terminology on bases or environments.

Definition 4 (Bases). *Let $\Gamma = x_1{\prec}\, A_1, \ldots, x_n{\prec}\, A_n$ be a basis.*

- Γ *is called* legal *if* $\Gamma \vdash A : X$ *for some expression* A *and type* X.
- A *declaration* $x \prec A$ is in Γ, *notation* $x \prec A \in \Gamma$, *if* $x \equiv x_i$ *and* $A \equiv A_i$ *for some* i.
- Δ *is* part *of* Γ, *notation* $\Delta \subseteq \Gamma$, *if* $\Delta = x_{i_1} \prec A_{i_1}, \ldots, x_{i_k} \prec A_{i_k}$ *with* $1 \leq i_1 < \ldots < i_k \leq n$. *Note that the order is preserved.*
- Δ *is an* initial segment *of* Γ, *if* $\Delta = x_1 \prec A_1, \ldots, x_j \prec A_j$ *for some* $1 \leq j \leq n$.

The following lemma collects a number of simple properties of a type judgment. It states that if $\Gamma \vdash A : X$, then the elements of each multiset of X and variables of A is in domain of Γ. It also shows some relations among multisets of A and any legal basis always has distinct declarations.

Lemma 1 (Legal typing). *If* $\Gamma \vdash A : X$ *and* $c \in \mathbb{C}$, *then*

1. *elements of* $FV(A)$, X^i, X^o, X^j *and* X^p *are in* $Dom(\Gamma)$
2. $\Gamma \vdash \epsilon : \langle [\,], [\,], [\,], [\,] \rangle$
3. *every variable in* $Dom(\Gamma)$ *is declared only once in* Γ
4. $X^o(c) \leq X^i(c) \leq \mathcal{R}(c), X^p(c) \leq X^j(c) \leq \mathcal{R}(c)$
5. $0 \leq X^i(c) - X^j(c), X^o(c) - X^p(c) \leq 1$.

The following lemma is important in that it allows us to find a syntax-directed derivation of the type of an expression and hence it allows us to calculate the types of sub-expressions. This lemma is sometimes called the *inversion lemma of the typing relation* [9]. Note that in the third clause the sequential decomposition in A and B may not be unique.

Lemma 2 (Generation).

1. *If* $\Gamma \vdash \texttt{new}\, x : X$, *then* $x \in X^p$ *and there exists bases* Δ, Δ' *and expression* A *such that* $\Gamma = \Delta, x \prec A, \Delta'$, *and* $\Delta \vdash A : \langle X^i - x, X^o - x, X^j - x, X^p - x \rangle$.
2. *If* $\Gamma \vdash \texttt{reu}\, x : X$, *then* $x \in X^o$ *and there exists bases* Δ, Δ' *and expression* A *such that* $\Gamma = \Delta, x \prec A, \Delta'$, *and* $\Delta \vdash A : \langle X^i - x, X^o - x, X^j, X^p \rangle$.
3. *If* $\Gamma \vdash AB : Z$ *with* $A, B \neq \epsilon$, *then there exists* X, Y *such that* $\Gamma \vdash A : X$, $\Gamma \vdash B : Y$ *and* $Z = \langle X^i \cup (X^o \uplus Y^j) \cup Y^i, (X^o \uplus Y^p) \cup Y^o, X^j \cup (X^p \uplus Y^j), X^p \uplus Y^p \rangle$.
4. *If* $\Gamma \vdash (A + B) : Z$, *then there exists* X, Y *such that* $\Gamma \vdash A : X$, $\Gamma \vdash B : Y$ *and* $Z = X \cup Y$.
5. *If* $\Gamma \vdash \{A\} : \langle X^i, [\,], X^j, [\,] \rangle$, *then there exists multisets* X^o *and* X^p *such that* $\Gamma \vdash A : \langle X^i, X^o, X^j, X^p \rangle$.

The next lemma stresses the significance of the order of declarations in a legal basis in our type system. The initial segment Δ of a legal basis Γ is a legal basis for the expression of the consecutive declaration after Δ. Besides, because of the weakening rule, there can be many legal bases under which a well-typed expression can be derived.

Lemma 3 (Legal monotonicity).

1. *If* $\Gamma = \Delta, x \prec E, \Delta'$ *is legal, then* $\Delta \vdash E : X$ *for some* X.
2. *If* $\Gamma \vdash E : X$, $\Gamma \subseteq \Gamma'$ *and* Γ' *is legal, then* $\Gamma' \vdash E : X$.

The following lemma can be viewed as the inverse of the previous legal monotonicity lemma. Under certain conditions we can contract a legal basis so that the expression is still well-typed in the new basis.

Lemma 4 (Strengthening). *If $\Gamma, x{\prec\!\!\!-} A \vdash B : Y$ and $x \notin FV(B)$, then $\Gamma \vdash B : Y$ and $x \notin Y^i$.*

In our type system, when an expression has a type this type is unique. This property is stated in the following proposition.

Proposition 1 (Uniqueness of types). *If $\Gamma \vdash A : X$ and $\Gamma \vdash A : Y$, then $X^i = Y^i$, $X^o = Y^o$, $X^j = Y^j$ and $X^p = Y^p$.*

Now we are going to state an important invariant of our operational semantics. We will use an approach inspired by Wright and Felleisen [12]. As our safety property involves the number of instances for every components in the stack, we define the notion of *well-typed configuration*, which includes both the well-typedness of expressions in the configuration and safety of the whole stack. Then we will state that the well-typedness of configurations is preserved during transitions.

Definition 5 (Well-typed configuration). *Let Γ be a legal basis. Configuration $ST = (M_1, E_1) \circ \cdots \circ (M_n, E_n)$ is well-typed with respect to requirement $\mathcal{R}$, notation $\Gamma, ST \models \mathcal{R}$, if it satisfies the followings:*

1. *$\Gamma \vdash_R E_k : X_k$ for all $1 \leq k \leq n$*
2. *$(X_k^j \uplus \biguplus_{l=1}^{k} M_l)(c) \leq \mathcal{R}(c)$ for all $1 \leq k \leq n$.*

Theorem 1 (Preservation). *If $\Gamma, ST_1 \models \mathcal{R}$ and $ST_1 \rightsquigarrow ST_2$, then $\Gamma, ST_2 \models \mathcal{R}$.*

The following corollary allows us to safely execute well-typed component programs. That is, during the execution of the programs the number of active instances of any component never exceeds the allowed number.

Corollary 1 (Safety). *Let $\Gamma \vdash_{\mathcal{R}} A : X$. Then for every sequence of transitions $([\,], A) \rightsquigarrow^* ST$ we have $[ST](c) \leq \mathcal{R}(c)$ for all c.*

Proof. Follows from Theorem 1, since $([\,], A)$ is a well-typed configuration.

5 Type Inference

So far we know that a well-typed program is safe to execute. Now, given a well-formed program, how do we know it is well-typed and how do we find the type of its starting expression? The problem of finding the type of an expression, given a set of declarations, is called the *type inference* problem [3] or *typability* problem [1]. Solving this problem relieves the programmer from giving the types explicitly and having them checked. The types inferred give also information

about the resources (e.g., memory) a component program uses, and can hence guide the design of the component system.

One may argue that we can test the behaviour of a component program by executing all possible runs under our operational semantics. However, this process could be *exponential* or even non-terminating (in the case of unforeseen circular dependencies.)

Now let us see a *polynomial* solution for our type inference problem. Let *Prog* be the component program and E be the expression we need to find the type of. A necessary (but not sufficient) condition is that the declarations in *Prog* can be reordered into a basis Γ such that for any declaration $x \prec A$ in Γ, the variables occurring in A are already declared previously in Γ. In other words:

$$\text{if } \Gamma = \Delta, x \prec A, \Delta' \text{ then } FV(A) \subseteq Dom(\Delta) \tag{1}$$

Such a reordering (if one exists) can be computed in polynomial time by an analysis of the dependency graph associated with the declarations in *Prog*. From now on we assume that Γ is a basis consisting of all declarations in *Prog* and satisfying (1). The considerations below are independent of which particular ordering is used as long as it satisfies (1).

The next step is that we reduce the problem of inferring a type for a given expression E to finding the types of $\texttt{new}\,x$ and $\texttt{reu}\,x$ for all $x \in FV(E)$. First write E as a sequential composition $E_1 \ldots E_p$ for some $p > 0$ in such a way that every E_i is of one of the following forms: $\texttt{new}\,x$, $\texttt{reu}\,x$, $(Exp + Exp)$, $\{Exp\}$. The type of E (if any) can then be inferrred from the types of the E_i's by the rule Seq in Definition 3. For the last two forms we have to apply the procedure recursively to the subexpressions, after which the type of the E_i in question can be inferred by the rules Choice and Scope, respectively. The type inference problem for E thus reduces to type inference problems for expressions $\texttt{new}\,x$ and $\texttt{reu}\,x$.

Finally we systematically infer types for $\texttt{new}\,x$ and $\texttt{reu}\,x$ for all $x \in \Gamma$, starting at the left. For the primitive components this is trivial. If we meet a declaration $Var \prec Exp$, then we know that all variables $x \in FV(Exp)$ are declared previously and we know the types of $\texttt{new}\,x$ and $\texttt{reu}\,x$. From these types we can infer the type of *Exp* and hence of $\texttt{new}\ Var$ and $\texttt{reu}\ Var$. Storing all the types inferred underway keeps type inference polynomial.

If at some place the above type inference procedure breaks down because the side condition of the rule Seq is violated, then there is no type. Here we have taken the strict approach where *all* declarations in the program have to be well-typed, even if they do not play a role, neither directly nor indirectly, in the expression which sparks off the execution of the program. For a top-down type inference procedure, where only the part of the program which is relevant for the expression is taken into consideration, see [8]. This is in many cases more efficient, but may hide ill-typed parts of the program that cause trouble in a later stage of the design.

6 Conclusions and Future Research

We have defined an abstract component language, an operational semantics and a type system such that every well-typed program can be executed safely in the sense that the number of simultaneously active instances never exceeds the maximum set for each component.

In a slightly more liberal approach one leaves out the side condition from the typing rule Seq and takes the types as counting the maximum number of simultaneously active instances of each component. These maxima can then be compared to the available resources.

We are well aware of the level of abstraction of the component language and plan to incorporate more language features. These include parallel composition and various forms of communication.

References

1. H. Barendregt. Lambda Calculi with Types. In: Abramsky, Gabbay, Maibaum (Eds.), *Handbook of Logic in Computer Science*, Vol. II. Oxford University Press. 1992.
2. M. Bezem and H. Truong. A Type System for the Safe Instantiation of Components, In *Proceedings of FOCLASA'03*, Electronic Notes in Theoretical Computer Science, September 2003.
3. L. Cardelli. Type systems. In A. B. Tucker, editor, *The Computer Science and Engineering Handbook*, chapter 103, pages 2208-2236. CRC Press, 1997.
4. K. Crary, D. Walker, and G. Morrisett. Typed Memory Management in a Calculus of Capabilities. In *Twenty-Sixth ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 262-275, San Antonio, TX, USA, January 1999.
5. K. Crary and S. Weirich. Resource Bound Certification. In *the Twenty-Seventh ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 184-198, Boston, MA, USA, January 2000.
6. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns - Elements of Reusable ObjectOriented Software*, Addison-Wesley, Reading, Mass., ISBN 0201633612, 1994.
7. E. Meijer and C. Szyperski. Overcoming Independent Extensibility Challenges, *Communications of the ACM*, Vol. 45, No. 10, pp. 41–44, October 2002.
8. H. Nilsen. *An Implementation of a Typing System for Counting Instances of Software Components*, MSc Thesis, Department of Informatics, University of Bergen, January 2005.
9. B. Pierce. *Types and Programming Languages*. MIT Press, ISBN 0262162091, February 2002.
10. J. C. Seco, Adding Type Safety to Component Programming, In *Proceedings of The PhD Student's Workshop* in FMOODS'02, University of Twente, the Netherlands, March 2002.
11. C. Szyperski. *Component Software: Beyond Object-Oriented Programming*, 2nd edition, Addison-Wesley, ISBN 0201745720, 2002.
12. Andrew K. Wright and Matthias Felleisen, A Syntactic Approach to Type Soundness, *Information and Computation*, Vol. 115, No. 1, pp. 38–94, 1994.

13. M. Zenger, Type-Safe Prototype-Based Component Evolution, *Proceedings of the European Conference on Object-Oriented Programming*, Malaga, Spain, June 2002.
14. M. Zenger, Programming Language Abstractions for Extensible Software Components, PhD Thesis, No. 2930, EPFL, Switzerland, March 2004.

A Proofs

In the sequel we use X^* for any of X^i, X^o, X^j and X^p.

Lemma 1 (Legal typing). *If $\Gamma \vdash A : X$ and $c \in \mathbb{C}$, then*

1. *elements of $FV(A)$, X^i, X^o, X^j and X^p are in $Dom(\Gamma)$*
2. *$\Gamma \vdash \epsilon : \langle [\,], [\,], [\,], [\,] \rangle$*
3. *every variable in $Dom(\Gamma)$ is declared only once in Γ*
4. *$X^o(c) \leq X^i(c) \leq \mathcal{R}(c), X^p(c) \leq X^j(c) \leq \mathcal{R}(c)$*
5. *$0 \leq X^i(c) - X^j(c), X^o(c) - X^p(c) \leq 1$.*

Proof. By simultaneous induction on derivation. Recall that Lemma 1 has 5 clauses.

- Base case Axiom, $\vdash \epsilon : \langle [\,], [\,], [\,], [\,] \rangle$ is trivial as $FV(\epsilon), X^*, Dom()$ are empty.
- Case Weaken,

$$(\mathsf{Weaken})\frac{\Gamma' \vdash A : X \quad \Gamma' \vdash B : Y \quad x \notin Dom(\Gamma')}{\Gamma', x \prec B \vdash A : X}$$

  Clause 3 follows by the side condition. The remaining clauses follow by IH.
- Case New,

$$(\mathsf{New})\frac{\Gamma' \vdash B : Y \quad x \notin Dom(\Gamma')}{\Gamma', x \prec B \vdash \texttt{new}\, x : \langle Y^i + x, Y^o + x, Y^j + x, Y^p + x \rangle}$$

  with $\Gamma = \Gamma', x \prec B$, $X^* = Y^* + x$. Assume the lemma is correct for the premise of this rule, so elements of $FV(B)$, Y^* are in $Dom(\Gamma')$. Clause 1 holds easily as the new element x in $FV(\texttt{new}\, x)$ and X^* is in $Dom(\Gamma) = Dom(\Gamma', x \prec B) = Dom(\Gamma') \cup x$. Clause 2 $\Gamma', x \prec B \vdash \epsilon : \langle [\,], [\,], [\,], [\,] \rangle$ follows by applying Weaken:

$$(\mathsf{Weaken})\frac{\Gamma' \vdash \epsilon : \langle [\,], [\,], [\,], [\,] \rangle \quad \Gamma' \vdash B : Y \quad x \notin Dom(\Gamma')}{\Gamma', x \prec B \vdash \epsilon : \langle [\,], [\,], [\,], [\,] \rangle}$$

  Clause 3 follows by the side condition $x \notin Dom(\Gamma')$. Clause 4 follows by IH. The last clause 5 holds since $x \notin Y^i$ and $x \notin Y^j$.
- Case Reu,

$$(\mathsf{Reu})\frac{\Gamma' \vdash B : Y \quad x \notin Dom(\Gamma')}{\Gamma, x \prec B \vdash \texttt{reu}\, x : \langle Y^i + x, Y^o + x, Y^j, Y^p \rangle}$$

  with $\Gamma = \Gamma', x \prec B$, $X = \langle Y^i + x, Y^o + x, Y^j, Y^p \rangle$. The proof is analogous to case New.

– Case Seq,

$$(\mathsf{Seq})\frac{\Gamma \vdash B : Y \quad \Gamma \vdash C : Z \quad (Y^o \uplus Z^j)(c) \leq \mathcal{R}(c) \quad B, C \neq \epsilon}{\Gamma \vdash BC : \langle Y^i \cup (Y^o \uplus Z^j) \cup Z^i, (Y^o \uplus Z^p) \cup Z^o, Y^j \cup (Y^p \uplus Z^j), Y^p \uplus Z^p \rangle}$$

Clauses 1, 2 and 3 hold by IH. For clause 4 it is to see that: $((Y^o \uplus Z^p) \cup Z^o)(c) \leq (Y^i \cup (Y^o \uplus Z^j) \cup Z^i)(c) \leq \mathcal{R}(c)$ holds since $Z^p \subseteq Z^j$, $Z^o \subseteq Z^i$ and $(Y^o \uplus Z^j)(c) \leq \mathcal{R}(c)$ from side condition, $Y^i, Z^i \leq \mathcal{R}(c)$ by IH. Similarly, $(Y^p \uplus Z^p)(c) \leq (Y^j \cup (Y^p \uplus Z^j))(c) \leq \mathcal{R}(c)$ holds since $Z^p \subseteq Z^j$ and $(Y^p \uplus Z^j)(c) \leq (Y^o \uplus Z^j)(c) \leq \mathcal{R}(c)$. For clause 5, as $Y^i(c) \geq Y^j(c)$ and $Z^o(c) \geq Z^p(c)$ for all c, we get $0 \leq X^i(c) - X^j(c)$ immediately. In addition,

$$X^i(c) - X^j(c) = max \left\{ \begin{array}{l} Y^i(c) - (Y^j \cup (Y^p \uplus Z^j))(c), \\ (Y^o \uplus Z^j)(c) - (Y^j \cup (Y^p \uplus Z^j))(c), \\ Z^i(c) - (Y^j \cup (Y^p \uplus Z^j))(c) \end{array} \right\}$$

each of the three cases is less then or equals 1 so $X^i(c) - X^j(c) \leq 1$. Similarly, it is easy to see that $0 \leq X^o(c) - X^p(c) = (Y^o \uplus Z^p) \cup Z^o)(c) - (Y^p \uplus Z^p)(c) \leq 1$.

– Case Choice,

$$(\mathsf{Choice})\frac{\Gamma \vdash C : Z \quad \Gamma \vdash B : Y}{\Gamma \vdash (C + B) : Z \cup Y}$$

Analogous to case Seq. First three clauses are easy.
Clause 4 holds since $max(Z^o(c), Y^o(c)) \leq \mathcal{R}(c)$ and $max(Z^i(c), Y^i(c)) \leq \mathcal{R}(c)$ by IH. Clause 5 is also easy:

$$0 \leq X^i(c) - X^j(c) = (Y^i \cup Z^i)(c) - (Y^j \cup Z^j)(c)$$

If $Y^i \supseteq Z^i$ then $X^i(c) - X^j(c) = Y^i(c) - (Y^j \cup Z^j)(c) \leq Y^i(c) - Y^j(c) \leq 1$. The other way around $Z^i \supseteq Y^i$ is the same. The second part, $0 \leq X^o(c) - X^p(c) \leq 1$, is analogous.

– Case Scope,

$$(\mathsf{Scope})\frac{\Gamma \vdash B : Y}{\Gamma \vdash \{B\} : \langle Y^i, [\,], Y^j, [\,] \rangle}$$

All clauses hold by IH or are trivial.

□

Lemma 2 (Generation).

1. *If $\Gamma \vdash \texttt{new}\, x : X$, then $x \in X^p$ and there exists bases Δ, Δ' and expression A such that $\Gamma = \Delta, x \prec A, \Delta'$, and $\Delta \vdash A : \langle X^i - x, X^o - x, X^j - x, X^p - x \rangle$.*
2. *If $\Gamma \vdash \texttt{reu}\, x : X$, then $x \in X^o$ and there exists bases Δ, Δ' and expression A such that $\Gamma = \Delta, x \prec A, \Delta'$, and $\Delta \vdash A : \langle X^i - x, X^o - x, X^j, X^p \rangle$.*
3. *If $\Gamma \vdash AB : Z$ with $A, B \neq \epsilon$, then there exists X, Y such that $\Gamma \vdash A : X$, $\Gamma \vdash B : Y$ and $Z = \langle X^i \cup (X^o \uplus Y^j) \cup Y^i, (X^o \uplus Y^p) \cup Y^o, X^j \cup (X^p \uplus Y^j), X^p \uplus Y^p \rangle$.*
4. *If $\Gamma \vdash (A + B) : Z$, then there exists X, Y such that $\Gamma \vdash A : X$, $\Gamma \vdash B : Y$ and $Z = X \cup Y$.*
5. *If $\Gamma \vdash \{A\} : \langle X^i, [\,], X^j, [\,] \rangle$, then there exists multisets X^o and X^p such that $\Gamma \vdash A : \langle X^i, X^o, X^j, X^p \rangle$.*

Proof. By induction on derivation.

1. $\Gamma \vdash \texttt{new}\, x : X$ can only be derived by the rule New or Weaken. If it is derived by the rule New, then there is only one possibility:

$$(\mathsf{New})\frac{\Delta \vdash A : Y \quad x \notin Dom(\Delta)}{\Delta, x \prec A \vdash \texttt{new}\, x : X}$$

with $X^* = Y^* + x$ and $\Gamma = \Delta, x \prec A$, so that Δ' is empty.
If $\Gamma \vdash \texttt{new}\, x : X$ is derived by the rule Weaken:

$$(\mathsf{Weaken})\frac{\Gamma' \vdash \texttt{new}\, x : X \quad \Gamma' \vdash B : Y \quad y \notin Dom(\Gamma')}{\Gamma', y \prec B \vdash \texttt{new}\, x : X}$$

then $\Gamma' \vdash \texttt{new}\, x : X$ and by the IH applied to $\Gamma' \vdash \texttt{new}\, x : X$ we have $\Gamma' = \Delta_1, x \prec A, \Delta_2$ and $\Delta_1 \vdash A : \langle X^i - x, X^o - x, X^j - x, X^p - x \rangle$ for some Δ_1, Δ_2, and A. With $\Delta = \Delta_1$, $\Delta' = \Delta_2, y \prec B$ we have all the conclusions.
2. Case $\Gamma \vdash \texttt{reu}\, x : X$: analogous to clause 1.
3. $\Gamma \vdash AB : Z$ with $A, B \neq \epsilon$ can only be derived by the rule Seq or the rule Weaken. If $\Gamma \vdash AB : Z$ is derived by the rule Seq with two component expressions A and B in the premise of the typing rule:

$$(\mathsf{Seq})\frac{\Gamma \vdash A : X \quad \Gamma \vdash B : Y \quad (X^o \uplus Y^j)(c) \leq \mathcal{R}(c) \quad A, B \neq \epsilon}{\Gamma \vdash AB : \langle X^i \cup (X^o \uplus Y^j) \cup Y^i, (X^o \uplus Y^p) \cup Y^o, X^j \cup (X^p \uplus Y^j), X^p \uplus Y^p \rangle}$$

then the proof is immediate.
If $\Gamma \vdash AB : Z$ is derived by the rule Seq with two component expressions $A_1 \neq A$ and $B_1 \neq B$ such that $A_1 B_1 = AB$:

$$(\mathsf{Seq})\frac{\Gamma \vdash A_1 : X_1 \quad \Gamma \vdash B_1 : Y_1 \quad (X_1^o \uplus Y_1^j)(c) \leq \mathcal{R}(c) \quad A_1, B_1 \neq \epsilon}{\Gamma \vdash A_1 B_1 : \langle X_1^i \cup (X_1^o \uplus Y_1^j) \cup Y_1^i, (X_1^o \uplus Y_1^p) \cup Y_1^o, X_1^j \cup (X_1^p \uplus Y_1^j), X_1^p \uplus Y_1^p \rangle}$$

then there are two possibilities:
   - $A = A_1 A_2$: then $B_1 = A_2 B$ and we have $\Gamma \vdash A_2 B : Y_1$.
     By the IH applied to $\Gamma \vdash A_2 B : Y_1$ we get $\Gamma \vdash A_2 : X_2$ and $\Gamma \vdash B : Y$ with $X = \langle X_2^i \cup (X_2^o \uplus Y^j) \cup Y^i, (X_2^o \uplus Y^p) \cup Y^o, X_2^j \cup (X_2^p \uplus Y^j), X_2^p \uplus Y^p \rangle$. As the side condition $(X_1^o \uplus X_2^j)(c) \leq (X_1^o \uplus (X_2^j \cup (X_2^p \uplus Y^j)))(c) = (X_1^o \uplus Y_1^j)(c) \leq \mathcal{R}(c)$ holds, we can apply the rule Seq to $\Gamma \vdash A_1 : X_1$ and $\Gamma \vdash A_2 : X_2$ and get $\Gamma \vdash A : X$ with $X = \langle X_1^i \cup (X_1^o \uplus X_2^j) \cup X_2^i, (X_1^o \uplus X_2^p) \cup X_2^o, X_1^j \cup (X_1^p \uplus X_2^j), X_1^p \uplus X_2^p \rangle$. We still need to show that $Z = \langle X^i \cup (X^o \uplus Y^j) \cup Y^i, (X^o \uplus Y^p) \cup Y^o, X^j \cup (X^p \uplus Y^j), X^p \uplus Y^p \rangle$, that is we need to prove four equations:

$$\begin{array}{l} X^i \cup (X^o \uplus Y^j) \cup Y^i = X_1^i \cup (X_1^o \uplus Y_1^j) \cup Y_1^i, \\ (X^o \uplus Y^p) \cup Y^o = (X_1^o \uplus Y_1^p) \cup Y_1^o, \\ X^j \cup (X^p \uplus Y^j) = X_1^j \cup (X_1^p \uplus Y_1^j), \\ X^p \uplus Y^p = X_1^p \uplus Y_1^p \end{array}$$

     We have:

$$\begin{array}{l} X^i \cup (X^o \uplus Y^j) \cup Y^i \\ = (X_1^i \cup (X_1^o \uplus X_2^j) \cup X_2^i) \cup (((X_1^o \uplus X_2^p) \cup X_2^o) \uplus Y^j) \cup Y^i \\ = X_1^i \cup X_2^i \cup Y^i \cup (X_1^o \uplus X_2^j) \cup (((X_1^o \uplus X_2^p) \cup X_2^o) \uplus Y^j) \\ = X_1^i \cup X_2^i \cup Y^i \cup (X_1^o \uplus X_2^j) \cup (X_1^o \uplus X_2^p \uplus Y^j) \cup (X_2^o \uplus Y^j) \\ = X_1^i \cup X_2^i \cup Y^i \cup (X_1^o \uplus (X_2^j \cup (X_2^p \uplus Y^j))) \cup (X_2^o \uplus Y^j) \\ = X_1^i \cup (X_1^o \uplus (X_2^j \cup (X_2^p \uplus Y^j))) \cup (X_2^i \cup (X_2^o \uplus Y^j) \cup Y^i) \\ = X_1^i \cup (X_1^o \uplus Y_1^j) \cup Y_1^i \end{array}$$

so the first equation holds. Similarly,

$$\begin{aligned}
&(X^o \uplus Y^p) \cup Y^o \\
&= (((X_1^o \uplus X_2^p) \cup X_2^o) \uplus Y^p) \cup Y^o \\
&= (X_1^o \uplus X_2^p \uplus Y^p) \cup (X_2^o \uplus Y^p) \cup Y^o \\
&= (X_1^o \uplus (X_2^p \uplus Y^p)) \cup ((X_2^o \uplus Y^p) \cup Y^o) \\
&= (X_1^o \uplus Y_1^p) \cup Y_1^o
\end{aligned}$$

so the second equation holds.

$$\begin{aligned}
&X^j \cup (X^p \uplus Y^j) \\
&= (X_1^j \cup (X_1^p \uplus X_2^j)) \cup ((X_1^p \uplus X_2^p) \uplus Y^j) \\
&= X_1^j \cup (X_1^p \uplus X_2^j) \cup (X_1^p \uplus X_2^p \uplus Y^j) \\
&= X_1^j \cup (X_1^p \uplus (X_2^j \cup (X_2^p \uplus Y^j))) \\
&= X_1^j \cup (X_1^p \uplus Y_1^j)
\end{aligned}$$

so the third equation holds. The last equation follows easily:

$$X^p \uplus Y^p = (X_1^p \uplus X_2^p) \uplus Y^p = X_1^p \uplus (X_2^p \uplus Y^p) = X_1^p \uplus Y_1^p \ .$$

- $B = B_0 B_1$: then $A_1 = AB_0$. By analogous reasoning as in the previous case we get the conclusions.

If $\Gamma \vdash AB : Z$ is derived by the rule Weaken:

$$(\mathsf{Weaken})\frac{\Gamma' \vdash AB : Z \quad \Gamma' \vdash C : V \quad y \notin Dom(\Gamma')}{\Gamma', y \prec C \vdash AB : Z}$$

with $\Gamma = \Gamma', y \prec C$ then by the IH applied to $\Gamma' \vdash AB : Z$ we have $\Gamma' \vdash A : X$, $\Gamma' \vdash B : Y$, $Z = \langle X^i \cup (X^o \uplus Y^j) \cup Y^i, (X^o \uplus Y^p) \cup Y^o, X^j \cup (X^p \uplus Y^j), X^p \uplus Y^p \rangle$. Now weakening $\Gamma' \vdash A : X$ and $\Gamma' \vdash B : Y$ to $\Gamma = \Gamma', y \prec C$ we have all the conclusions.

4. $\Gamma \vdash (A + B) : Z$ can only be derived by the rule Choice or the rule Weaken. If it is derived by the rule Choice, then there is only one possibility:

$$(\mathsf{Choice})\frac{\Gamma \vdash A : X \quad \Gamma \vdash B : Y}{\Gamma \vdash (A + B) : X \cup Y}$$

with $Z = X \cup Y$. The conclusion follows immediately.
If $\Gamma \vdash (A + B) : Z$ is derived by the rule Weaken:

$$(\mathsf{Weaken})\frac{\Gamma' \vdash (A + B) : Z \quad \Gamma' \vdash E : V \quad x \notin Dom(\Gamma')}{\Gamma', x \prec E \vdash (A + B) : Z}$$

then the proof is analogous to the proof of case Weaken in the previous clause.

5. $\Gamma \vdash \{A\} : \langle X^i, [\,], X^j, [\,] \rangle$ can only be derived by the rule Scope or the rule Weaken. The proof is analogous to the proof of the previous clause.

□

Lemma 3 (Legal monotonicity).

1. *If* $\Gamma = \Delta, x \prec E, \Delta'$ *is legal, then* $\Delta \vdash E : X$ *for some* X.
2. *If* $\Gamma \vdash E : X$, $\Gamma \subseteq \Gamma'$ *and* Γ' *is legal, then* $\Gamma' \vdash E : X$.

Proof. 1. The only way to extend Δ to $\Delta, x \prec E$ in a derivation is by applying the rule New, Reu or Weaken.

$$(\mathsf{New})\frac{\Gamma \vdash E : X \quad x \notin Dom(\Gamma)}{\Gamma, x \prec E \vdash \mathtt{new}\, x : \langle X^i + x, X^o + x, X^j + x, X^p + x\rangle}$$

$$(\mathsf{Reu})\frac{\Gamma \vdash E : X \quad x \notin Dom(\Gamma)}{\Gamma, x \prec E \vdash \mathtt{reu}\, x : \langle X^i + x, X^o + x, X^j, X^p\rangle}$$

$$(\mathsf{Weaken})\frac{\Gamma \vdash E : X \quad \Gamma \vdash B : Y \quad x \notin Dom(\Gamma)}{\Gamma, x \prec E \vdash B : Y}$$

Each of the rules has $\Delta \vdash E : X$ as a premise.

2. By induction on derivation of $\Gamma \vdash E : X$. We prove that for all Γ' legal such that $\Gamma \subseteq \Gamma'$ we have $\Gamma' \vdash E : X$.
   - Base case Axiom, $E = \epsilon$, then $\Gamma' \vdash \epsilon : \langle [\,], [\,], [\,], [\,]\rangle$ since Γ' is legal.
   - Case New, $E = \mathtt{new}\, x$

     $$(\mathsf{New})\frac{\Delta \vdash B : Y \quad x \notin Dom(\Delta)}{\Delta, x \prec B \vdash \mathtt{new}\, x : X}$$

     with $X = \langle Y^i + x, Y^o + x, X^i + x, X^o + x\rangle$ and $\Gamma = \Delta, x \prec B$. Because $\Gamma \subseteq \Gamma'$ with Γ' legal there exists Δ_1, Δ_2 such that $\Delta \subseteq \Delta_1$ and $\Delta_1, x \prec B, \Delta_2 = \Gamma'$, with all initial segments of Γ' are legal. By clause 1 we have $\Delta_1 \vdash B : Y$. As x occurs only once in Γ' we have $x \notin Dom(\Delta_1)$ and we can apply the rule New to get $\Delta_1, x \prec B \vdash \mathtt{new}\, x : X$. Since Γ' is legal we can iterate the rule Weaken to get $\Gamma' \vdash \mathtt{new}\, x : X$.
   - Case Reu, $E = \mathtt{new}\, x$: analogous to case New.
   - Case Weaken,

     $$(\mathsf{Weaken})\frac{\Delta \vdash E : X \quad \Delta \vdash B : Y \quad x \notin Dom(\Delta)}{\Delta, x \prec B \vdash E : X}$$

     with $\Gamma = \Delta, x \prec B$. Because $\Gamma \subseteq \Gamma'$ and Γ' legal, we have $\Delta \subseteq \Gamma'$. By the IH we get immediately $\Gamma' \vdash E : X$.
   - Case Seq, $E = BC$ with $B, C \neq \epsilon$: by Generation Lemma we have $\Gamma \vdash B : Y$ and $\Gamma \vdash C : Z$. By the IH we have $\Gamma' \vdash B : Y$ and $\Gamma' \vdash C : Z$. As the side condition for $\Gamma \vdash BC : \langle Y^i \cup (Y^o \uplus Z^j) \cup Z^i, (Y^o \uplus Z^p) \cup Z^o, Y^j \cup (Y^p \uplus Z^j), Y^p \uplus Z^p\rangle$ holds we can apply the rule Seq we get the conclusion.
   - Case Choice, $E = (B + C)$: analogous to the case Seq.
   - Case Scope, $E = \{B\}$: analogous to the case Seq.

□

Lemma 4 (Strengthening). *If $\Gamma, x{\prec}A \vdash B : Y$ and $x \notin FV(B)$, then $\Gamma \vdash B : Y$ and $x \notin Y^i$.*

Proof. By induction on derivation. Let $\Gamma' = \Gamma, x{\prec}A$

- Case Axiom, $B = \epsilon$: does not apply since the basis is not empty.
- Case New, $B = \texttt{new}\, x$: does not apply since $FV(B) = FV(\texttt{new}\, x) = \{x\}$.
- Case Reu, $B = \texttt{reu}\, x$: does not apply since $FV(B) = FV(\texttt{reu}\, x) = \{x\}$.
- Case Weaken,
$$(\mathsf{Weaken})\frac{\Gamma \vdash A : X \quad \Gamma \vdash B : Y \quad x \notin Dom(\Gamma)}{\Gamma, x{\prec}A \vdash B : Y}$$
We have $\Gamma \vdash B : Y$ in the premise and $x \notin Y^i$ by IH.
- Case Seq, $B = B_1 B_2$:
$$(\mathsf{Seq})\frac{\Gamma' \vdash B_1 : Y_1 \quad \Gamma' \vdash B_2 : Y_2 \quad (Y_1^o \uplus Y_2^j)(c) \leq \mathcal{R}(c) \quad B_1, B_2 \neq \epsilon}{\Gamma' \vdash B_1 B_2 : \langle Y_1^i \cup (Y_1^o \uplus Y_2^j) \cup Y_2^i, (Y_1^o \uplus Y_2^p) \cup Y_2^o, Y_1^j \cup (Y_1^p \uplus Y_2^j), Y_1^p \uplus Y_2^p \rangle}$$
with $Y = \langle Y_1^i \cup (Y_1^o \uplus Y_2^j) \cup Y_2^i, (Y_1^o \uplus Y_2^p) \cup Y_2^o, Y_1^j \cup (Y_1^p \uplus Y_2^j), Y_1^p \uplus Y_2^p \rangle$. Since $x \notin FV(B_1 B_2) = FV(B_1) \cup FV(B_2)$ we have $x \neq FV(B_1)$ and $x \notin FV(B_2)$. By IH we get $\Gamma \vdash B_1 : Y_1$ and $x \notin Y_1^i$, $\Gamma \vdash B_2 : Y_2$ and $x \notin Y_2^i$. As the side condition does not change at all, we can apply the rule Seq to get the conclusion: $\Gamma \vdash B_1 B_2 : Y$.
- Case Choice, $B = (B_1 + B_2)$: analogous to the case Seq.
- Case Scope, $B = \{C\}$: analogous to the case Seq.

□

Proposition 1 (Uniqueness of types). *If $\Gamma \vdash A : X$ and $\Gamma \vdash A : Y$, then $X^i = Y^i$, $X^o = Y^o$, $X^j = Y^j$ and $X^p = Y^p$.*

Proof. By induction on the derivation of $\Gamma \vdash A : X$.

- Base case Axiom, we have $A = \epsilon$ and Γ is empty, so that only Axiom is applicable. Hence, $X = Y = \langle [], [], [], [] \rangle$.
- Case New,
$$(\mathsf{New})\frac{\Gamma' \vdash B : U \quad x \notin Dom(\Gamma)}{\Gamma', x{\prec}B \vdash \texttt{new}\, x : X}$$
with $X^* = U^* + x$ and $\Gamma = \Gamma', x{\prec}B$. Assume this Proposition 1 holds for the premise of this rule and let $\Gamma \vdash \texttt{new}\, x : Y$. By Generation Lemma 2, $x \in Y^*$, $\Gamma = \Delta_1, x{\prec}C, \Delta_2$ and $\Delta_1 \vdash C : \langle Y^i - x, Y^o - x, Y^j - x, Y^p - x \rangle$ for some Δ_1, Δ_2, C.
By Lemma 1, there is only one declaration of x in Γ. This means $\Delta_1 = \Gamma'$, $C = B$ and Δ_2 is empty, so $\Gamma' \vdash B : \langle Y^i - x, Y^o - x, Y^j - x, Y^p - x \rangle$. By IH we have $X^* - x = Y^* - x$, i.e. $X = Y$.
- Case Reu, analogous to case New.
- Case Weaken, let $\Gamma = \Gamma', x{\prec}B$ such that:
$$(\mathsf{Weaken})\frac{\Gamma' \vdash A : X \quad \Gamma' \vdash B : Z \quad x \notin Dom(\Gamma')}{\Gamma', x{\prec}B \vdash A : X}$$
Assume this Proposition 1 holds for the two premises and let $\Gamma \vdash A : Y$. Since $\Gamma' \vdash A : X$ and $x \notin Dom(\Gamma')$ we have $x \notin FV(A)$. By Lemma 4 applied to $\Gamma', x{\prec}B \vdash A : Y$ we get $\Gamma' \vdash A : Y$. By IH we have the conclusion $X = Y$.

– Case Seq, let $\Gamma \vdash B_1 B_2 : X$ with $B_1, B_2 \neq \epsilon$ such that:

$$(\mathsf{Seq})\frac{\Gamma \vdash B_1 : Y_1 \quad \Gamma \vdash B_2 : Y_2 \quad (Y_1^o \uplus Y_2^j)(c) \leq \mathcal{R}(c) \quad B_1, B_2 \neq \epsilon}{\Gamma \vdash B_1 B_2 : \langle Y_1^i \cup (Y_1^o \uplus Y_2^j) \cup Y_2^i, (Y_1^o \uplus Y_2^p) \cup Y_2^o, Y_1^j \cup (Y_1^p \uplus Y_2^j), Y_1^p \uplus Y_2^p \rangle}$$

By Generation Lemma 2 applied to $\Gamma \vdash B_1 B_2 : Y$ we have $\Gamma \vdash B_1 : V_1$, $\Gamma \vdash B_2 : V_2$, $Y = \langle V_1^i \cup (V_1^o \uplus V_2^j) \cup V_2^i, (V_1^o \uplus V_2^p) \cup V_2^o, V_1^j \cup (V_1^p \uplus V_2^j), V_1^p \uplus V_2^p \rangle$. By the IH, we have $Y_1 = V_1$ and $Y_2 = V_2$. Hence, $X = Y = \langle Y_1^i \cup (Y_1^o \uplus Y_2^j) \cup Y_2^i, (Y_1^o \uplus Y_2^p) \cup Y_2^o, Y_1^j \cup (Y_1^p \uplus Y_2^j), Y_1^p \uplus Y_2^p \rangle$.
– Case Choice, analogous to case Seq.
– Case Scope, analogous to case Seq. □

Theorem 1 (Preservation). *If $\Gamma, ST_1 \models \mathcal{R}$ and $ST_1 \rightsquigarrow ST_2$, then $\Gamma, ST_2 \models \mathcal{R}$.*

Proof. By case analysis on the all possible small-step transitions. In all cases we assume $\Gamma, ST_1 \models \mathcal{R}$ and $ST_1 \rightsquigarrow ST_2$, and we prove $\Gamma, ST_2 \models \mathcal{R}$. See Definition 5 for $\Gamma, ST_i \models \mathcal{R}$ (ST_i well-typed wrt. Γ and $\mathcal{R}$). See Table 1 for $\rightsquigarrow$. In most cases only the top of the stack is affected, in some cases the stack is pushed or popped. We restrict our attention to the parts of the stack that change.

– **Case osNew**,

$$\begin{array}{ll} (\textbf{osNew}) & \text{if } x \prec A \in \mathit{Prog} \\ ST_1 = ST \circ (M, \texttt{new}\, xE) & \rightsquigarrow ST \circ (M + x, AE) = ST_2 \end{array}$$

Since ST_1 is well-typed, there exists X such that $\Gamma \vdash \texttt{new}\, xE : X$ and $([ST] \uplus M \uplus X^j)(c) \leq \mathcal{R}(c)$ for all c. We only need to prove that $\Gamma \vdash AE : Z$ and $([ST] \uplus (M + x) \uplus Z^j)(c) \leq \mathcal{R}(c)$ for all c since the stack only changes at the top.
We prove AE well-typed as follows. From $\Gamma \vdash \texttt{new}\, xE : X$ we get, by Generation Lemma 2, $\Gamma \vdash \texttt{new}\, x : X_1$ and $\Gamma \vdash E : X_2$. Also by that lemma we have $\Gamma \vdash A : Y$ with $Y^* = X_1^* - x$. So we can derive $\Gamma \vdash AE : \langle Y^i \cup (Y^o \uplus X_2^j) \cup X_2^i, (Y^o \uplus X_2^p) \cup X_2^o, Y^j \cup (Y^p \uplus X_2^j), Y^p \uplus X_2^p \rangle$ by applying the rule Seq as the side condition $(Y^o \uplus X_2^j)(c) \leq \mathcal{R}(c)$ holds by $Y^o \subset X_1^o$ and $(X_1^o \uplus X_2^j)(c) \leq \mathcal{R}(c)$ for all c.
Next we prove that $([ST] \uplus (M + x) \uplus (Y^j \cup (Y^p \uplus X_2^j)))(c) \leq \mathcal{R}(c)$ for all c. We have:

$$\begin{array}{ll} LHS = ([ST] \uplus (M + x) \uplus ((X_1^j - x) \cup ((X_1^p - x) \uplus X_2^j)))(c) & \\ = ([ST] \uplus M \uplus (X_1^j \cup (X_1^p \uplus X_2^j)))(c) & \\ = ([ST] \uplus M \uplus X^j)(c) & \\ \leq \mathcal{R}(c) & \text{(as } ST_1 \text{ well-typed)} \end{array}$$

– **Case osReu1**,

$$\begin{array}{ll} (\textbf{osReu1}) & \text{if } x \prec A \in \mathit{Prog} \quad x \notin [ST] \uplus M \\ ST_1 = ST \circ (M, \texttt{reu}\, xE) & \rightsquigarrow ST \circ (M + x, AE) = ST_2 \end{array}$$

Since ST_1 is well-typed, there exists X such that $\Gamma \vdash \texttt{reu}\, xE : X$ and $([ST] \uplus M \uplus X^j)(c) \leq \mathcal{R}(c)$ for all c. As in case **osNew** we only need to prove that $\Gamma \vdash AE : Z$ and $([ST] \uplus (M + x) \uplus Z^j)(c) \leq \mathcal{R}(c)$ for all c.
First we prove that AE is well-typed. From $\Gamma \vdash \texttt{reu}\, xE : X$ we get, by Generation Lemma 2, $\Gamma \vdash \texttt{reu}\, x : X_1$ and $\Gamma \vdash E : X_2$. Also by that lemma we have $\Gamma \vdash A : Y$

with $Y = \langle X_1^i - x, X_1^o - x, X_1^j, X_1^p \rangle$. So we can derive $\Gamma \vdash AE : \langle Y^i \cup (Y^o \uplus X_2^j) \cup X_2^i, (Y^o \uplus X_2^p) \cup X_2^o, Y^j \cup (Y^p \uplus X_2^j), Y^p \uplus X_2^p \rangle$ as the side condition $(Y^o \uplus X_2^j)(c) \leq \mathcal{R}(c)$ holds by $Y^o \subseteq X_1^o$ and $(X_1^o \uplus X_2^j)(c) \leq \mathcal{R}(c)$ for all c.
Next we prove the second clause, $([ST] \uplus (M + x) \uplus Z^j)(c) \leq \mathcal{R}(c)$ for all c. Note that $Z^j = X_1^j \cup (X_1^p \uplus X_2^j) = X^j$ so we have $([ST] \uplus (M + x) \uplus Z^j)(c) \leq \mathcal{R}(c)$ holds for all $c \neq x$ by assumption. For $c = x$, as $x \notin [ST] \uplus M$ we have:

$$\begin{aligned} LHS &= (x + Z^j)(x) \\ &= (x + X^j)(x) \\ &= X^i(x) && \text{(Lemma 1, point 5, applied to Y)} \\ &\leq \mathcal{R}(x) = \mathcal{R}(c) && \text{(Lemma 1, point 4)} \end{aligned}$$

– **Case osReu2**,

$$\begin{array}{l} (\textbf{osReu2}) \quad \text{if } x \prec A \in \mathit{Prog} \;\; x \in [ST] \uplus M \\ ST_1 = ST \circ (M, \texttt{reu}\, xE) \rightsquigarrow ST \circ (M, AE) = ST_2 \end{array}$$

Since ST_1 is well-typed, there exists X such that $\Gamma \vdash \texttt{reu}\, xE : X$ and $([ST] \uplus M \uplus X^j)(c) \leq \mathcal{R}(c)$ for all c. As in case **osNew** we only need to prove that $\Gamma \vdash AE : Z$ and $(M \uplus [ST] \uplus Z^j)(c) \leq \mathcal{R}(c)$ for all c.
The proof of $\Gamma \vdash AE : Z$ is verbatim the same as in the previous case, with again $Z^j = X^j$, which trivializes the second proof obligation.

– **Case osChoice1**,

$$\begin{array}{l} (\textbf{osChoice1}) \\ ST_1 = ST \circ (M, (A + B)E) \rightsquigarrow ST \circ (M, AE) = ST_2 \end{array}$$

Since ST_1 is well-typed, there exists X such that $\Gamma \vdash (A+B)E : X$ and $([ST] \uplus M \uplus X^j)(c) \leq \mathcal{R}(c)$ for all c. We prove that $\Gamma \vdash AE : Z$ and $([ST] \uplus M \uplus Z^j)(c) \leq \mathcal{R}(c)$ for all c.
First we prove that AE is well-typed. By Generation Lemma 2 applied to $\Gamma \vdash (A + B)E : X$ we have $\Gamma \vdash (A + B) : X_1$, $\Gamma \vdash E : X_2$. Also by that lemma applied to $\Gamma \vdash (A + B) : X_1$ we have $\Gamma \vdash A : Y$ with $Y^* \subseteq X_1^*$. So we can derive $\Gamma \vdash AE : \langle Y^i \cup (Y^o \uplus X_2^j) \cup X_2^i, (Y^o \uplus X_2^p) \cup X_2^o, Y^j \cup (Y^p \uplus X_2^j), Y^p \uplus X_2^p \rangle$ as the side condition obviously holds.
Next we prove the second clause $([ST] \uplus M \uplus Z^j)(c) \leq \mathcal{R}(c)$ for all c. We have

$$\begin{aligned} LHS &= ([ST] \uplus M \uplus (Y^j \cup (Y^p \uplus X_2^j)))(c) \\ &\leq ([ST] \uplus M \uplus (X_1^j \cup (X_1^p \uplus X_2^j)))(c) \\ &= ([ST] \uplus M \uplus X^j)(c) \\ &\leq \mathcal{R}(c) && \text{(since } ST_1 \text{ well-typed)} \end{aligned}$$

– **Case osChoice2**, symmetric to case **osChoice1**.

– **Case osPush**,

$$\begin{array}{l} (\textbf{osPush}) \\ ST_1 = ST \circ (M, \{A\}E) \rightsquigarrow ST \circ (M, E) \circ ([\,], A) = ST_2 \end{array}$$

Since ST_1 is well-typed, there exists X such that $\Gamma \vdash \{A\}E : X$ and $([ST] \uplus M \uplus X^j)(c) \leq \mathcal{R}(c)$ for all c. Because we have pushed the stack, we have two things to check. For the well-typedness of the new configuration, we need to prove that (1)

$\Gamma \vdash A : Y$ and $([ST] \uplus M \uplus [] \uplus Y^j)(c) \leq \mathcal{R}(c)$ for all c, and (2) $\Gamma \vdash E : X_2$ and $([ST] \uplus M \uplus X_2^j)(c) \leq \mathcal{R}(c)$ for all c.
We prove (1) as follows. From $\Gamma \vdash \{A\}E : X$ we get, by Generation Lemma 2, $\Gamma \vdash \{A\} : X_1$ and $\Gamma \vdash E : X_2$ with $X^j = X_1^j \cup (X_1^p \uplus X_2^j)$. Also by that lemma we have $\Gamma \vdash A : Y$ with $Y = \langle X_1^i, Y^o, X_1^j, Y^p \rangle$. The second requirement follows from $Y^j = X_1^j \subseteq X^j$ and $([ST] \uplus M \uplus X^j)(c) \leq \mathcal{R}(c)$ for all c.
For (2) the first requirement is proved in (1). The second requirement follows from $X_2^j \subseteq X_1^p \uplus X_2^j \subseteq X^j$ and $([ST] \uplus M \uplus X^j)(c) \leq \mathcal{R}(c)$ for all c.

– **Case osPop**,

(**osPop**)
$ST \circ (M, E) \circ (M', \epsilon) \rightsquigarrow ST \circ (M, E)$

We immediately get $ST \circ (M, E)$ well-typed by $ST \circ (M, E) \circ (M', \epsilon)$ well-typed.

□